OPEN PROTOCOLS FOR WEB-BASED EDUCATIONAL MATERIALS

Terence Ahern[1], *Nancy Van Cleave*[2], *Brooks York*[3]

Abstract *— Using computers to deliver instructional material should be an excellent match of technology to education. Countless products meant to enhance or replace the classroom experience are currently available, yet few of them are being utilized in spite of the drive to increase the use of educational technology. Factors behind this under-utilization are complex, but include cost, lack of flexibility, lack of teacher training and support, and the proprietary nature of the products. Some companies in the electronic gaming industry have separated game content from the engine or delivery mechanism, thus saving development time and expense. Utilization of the Internet as an educational tool, especially for distance learning, is taken for granted. Thus, any computer-based instructional tool should have the following characteristics in order to have maximum utility at minimum expense: (1) be cross-platform, (2) separate content from delivery, (3) provide creation, editing, and delivery capabilities and (4) use open standards. Our work shows the feasibility of meeting all these requirements with XML and Java.*

Index Terms — Computers and Education, Educational Technology, Open Protocols, Web-based Education,

INTRODUCTION

Given the proliferation of expensive proprietary educational software, it is natural to ask whether an open protocol model would be feasible as an alternative, particularly for course authoring systems. Using computers to deliver instructional material should be an excellent match of technology to education since they, unlike textbooks, can provide interactive multimedia hypertext presentations. However, countless commercial products meant to enhance or replace the classroom experience are currently available, yet few of them are being utilized in spite of the drive to increase the use of educational technology [1].

Several factors account for this lack of utilization [2], including:

- cost of proprietary systems, individual CD's, site licenses, maintenance, upgrading, personnel, etc.
- lack of availability for particular configurations or machine types (notably, non-uniform legacy computers in disadvantaged and under-funded public schools)
- lack of flexibility in the software to tailor it to the specific and changing needs of a particular classroom, teacher, or student
- steep learning curves with little investment in teacher training and support
- lack of time and energy for instructors to invest in technology; discovering how to integrate technology into education
- limited classroom usefulness in general

In the next section we discuss the issues behind these problems and why they need to be taken into account when designing educational software, what choices are available for consideration, and the relative merits of these choices. We conclude that section with a list of requirements for educational software. In the sections following, we then describe how we fulfill these requirements with XML and Java, and thus prove the feasibility of our approach.

RELEVANT EDUCATIONAL & TECHNOLOGICAL ISSUES

Consideration of the problems surrounding the use of computers and other technology in education is important if we are to increase the use of technology in public classrooms, provide quality education to all students, and develop effective distance learning programs which may be utilized as readily by deployed military personnel as by corporate America. These issues cut across disciplines, having their roots in the fields of both Education and Technology. The concerns relevant to our research, owing to their impact on decisions related to solutions of the aforementioned problems, are presented below.

PROPRIETARY VS OPEN SYSTEMS

Several of the problems listed in the introduction stem from the proprietary nature of current educational products and software. When using a proprietary or closed-system design, only the group implementing the product, application, or system has total knowledge of the inner workings of the software and generally does not release these to the public at large. This prevents, or at least severely hampers, third party vendors from developing add-ons or extensions to the system.

[1] Terence Ahern, Texas Tech University, College of Education, Lubbock TX, dwtca@TTACS.TTU.EDU
[2] Nancy Van Cleave, Eastern IL University, Math Dept. 331 OM, 600 Lincoln Ave, Charleston, IL 61920, cfnkv@eiu.edu
[3] Brooks York, Austin, TX

An approach at the other end of the spectrum is to base the development of instructional tools on an open-system model where the protocols and data representations are published publicly and are thus available to all parties. This allows anyone to develop a tool that can utilize a specific protocol, which in turn may extend the usefulness of the system. Often these modifications and improvements are made available to other users with no charge or for an amount much less than similar proprietary software. Examples of these two stratagems are MS Windows (proprietary) and the Linux (open) operating systems.

In general, open systems provide less functionality and offer fewer administrative-type tools (especially in their early stages of development) than proprietary systems but usually are less expensive and require less server and system administration support [3].

SEPARATING CONTENT FROM DELIVERY

Some companies in the electronic gaming industry have separated game content from the game engine or delivery mechanism, thus saving development time and expense. New games are created by designing new content (creatures, weapons, scenarios), then adding it to the pre-existing presentation engine, which may be refined or updated to allow for new advances in technology. This separation of content from delivery system is an important concept since the reuse of code decreases development time and final cost.

In the educational realm, this sort of partitioning would allow technology-oriented people to develop and refine the authoring and presentation mechanisms, while educators could develop and refine the informational lesson content. Furthermore, educators would have the ability to share tutorials and other types of lessons, rather than everyone constructing their own from the ground up. It also makes modifying lessons presented in this format much easier since the educator has full control over the content and how it is ordered.

Rather than making this division, however, educational software is currently most often built as an integrated whole. The integration of data into educational software necessitates the entire rewriting of both content and engine since they are inseparable. As there is currently no such delivery engine, and since educational software in general does not generate a profit, no one is bothering to develop one.

WEB-BASED EDUCATION AND BROWSERS

With the frenzy over distance learning reaching fever pitch, use of the Internet as an educational tool is taken for granted. While it provides access to a vast amount of information, web-based instructional tools at all levels must still direct the learner through the material to some extent, otherwise the acquisition of content suffers as the student become mired in information overload [4]. Web-based instruction, then, must guide the learner through what the author has deemed the important concepts and information, rather than allowing the student free rein to roam the Internet.

A browser which does not have access to the Internet (or which is simply used as a stand-alone piece of software), may still utilize local files on either the user's computer or on the server to which it may be connected. Thus, the browser could be used to provide a platform from which to launch a presentation engine that orders the information and controls student access.

CREATING EDUCATIONAL PROTOCOLS

Major problems must be confronted when using an open-protocol model for educational applications. Programmers working within the traditional method of courseware development manage the sequencing of lesson content through the design of their proprietary delivery algorithms for how a specific user interacts and navigates the content. In other words, the software design itself controls access to the information when the data is an integral part of the software package. Because the classroom has many distinct instructional paradigms for content delivery (ranging from lecture to small-group discussion), the typical protocol development process is at a great disadvantage as many protocols (or one if it has an *extremely* versatile design) are required to represent the range of classroom instructional models.

Once a protocol is developed, it must also become part of the public domain by dissemination in order to be of wide use. What is needed then is a mechanism by which we not only create tutorials and other instructional aids which can be recognized, parsed, and delivered by the presentation engine, but also a way to associate the appropriate educational protocol with them.

INSTRUCTIONAL TOOLS REQUIREMENTS LIST

To address all of these problems, any computer-based instructional tool should satisfy the following requirements:

- utilize open standards and be accessible through the Internet
- be cross-platform and thus available on a wide variety of machines and configurations
- separate content from the delivery mechanism, allowing educators to control and share components
- provide creation, editing, and delivery mechanisms (preferably easy to learn, quick to use, and with a GUI).

In order to satisfy these constraints, the problem was broken down into three parts and the following solutions were then developed:

- an open protocol for tutorials based on the Document Type Definition in XML

- a Java application which incorporates the DTD in order to produce tutorial files consisting of information (course) content (such as filenames and URL's) supplied by the courseware author, and
- a Java applet which can be used to view the tutorial files created by the Java application.

The following sections discuss our solution choices in light of the above criteria, and give further details on their implementation.

OPEN STANDARDS

The first incarnation of the Internet began in the early 1970's with ARPANET. The original impetus behind the development of ARPANET was to solve the problem of allowing all the United States National Laboratories to continue to communicate and exchange data in the event one or more of the labs were destroyed in a nuclear war. Because these labs utilized different hardware, the communication between them was based on a single shared communication standard. These published interfaces (simply put, definitions of relevant data types and how they are stored) are crucial to this open interchange between various computers and users.

To integrate seamlessly with other Internet applications, educational software must take advantage of this openness and should itself utilize public interfaces. The openness of published communication standards encourages third party add-ons and free exchange of content. A prime example of how openness encourages system extensions is Linux. Many Linux components come out of Richard Stallman's GNU Project and the Free Software Foundation, which are open forums that encourage additional components for Linux.

DOCUMENT DEFINITION

Extensible Markup Language (XML) is an open standard for hypertext markup agreed upon by the World Wide Web Consortium. XML is based on Standard Generalized Markup Language (SGML), but is intended to be easier to use. The main reason XML is now preferred in some areas is because it is not limited to fixed element types like the hypertext markup language HTML. Instead, XML allows users to define their own hierarchy of data, which can be published (shared publicly) by using Document Type Definitions (DTDs). Once a DTD is created, it is possible to define custom sets of data that can work in the same way as other XML files already being used by an application.

XML solves the problem of characterizing relevant types of data files in an open environment through the use of DTDs, thus providing a protocol that may be shared across platforms. The DTD defines not only the structure of the document, but also the type and content of elements allowed in the document. A DTD can be carried by or associated with the document itself so the browser can properly determine the unique structure of the document.

A DTD can also stand alone as a separate file that defines a whole class of education-oriented documents. For example, we chose to model a tutorial in this work, and thus created a DTD which provides the framework or syntax rules common to all tutorial files. As long as tutorial documents based on this DTD are validated, i.e., properly follow the syntax rules contained in the DTD, a browser can determine how to access and parse them correctly.

A TUTORIAL PROTOCOL

One of the problems with proprietary commercial educational systems is that they adhere strictly to whatever instructional methodology paradigm the software designer chose to implement, rather than what the educator wishing to use the software might prefer. To avoid this problem, we elected to base our design on Gagne's Nine Events of Instruction [5], a general outline of the instructional design methodology for direct instruction that is commonly accepted. It includes:

(1) gaining attention
(2) informing the learner of the objective
(3) stimulating recall of prerequisite material
(4) presenting the stimulus material
(5) providing learning guidance
(6) eliciting the performance
(7) providing feedback about performance correctness
(8) assessing the performance
(9) enhancing retention and transfer

The DTD we designed, seen in Figure 1 below, specifies items 4, 6, and 8 in the authoring application [6]. The word *Syllabus* was chosen to designate a single tutorial, a group of tutorials for one course, or multiple groups of tutorials for several courses. Thus, a *Syllabus* is defined as a *Course*, which in turn is defined recursively as one or more *Courses* or *Units*. *Units* and *Lessons* are also defined recursively, so *Syllabi* may include many *Courses*, *Units* and *Lessons*. A *Syllabus*, then, can cover a single topic or organize the elements of an entire school year for an instructor.

```
<!--Syllabus DTD Version1.0-->
<!--Brooks York-->
<!ELEMENT Syllabus (Course)>
<!ELEMENT Course (name,(Course|Units)*)>
<!ELEMENT Units (name,(Units|Lesson)*,performance?)>
<!ELEMENT Lesson (name, (presentation|practice|performance|Lesson)*)>
<!ELEMENT name (#pcdata)>
<!ELEMENT presentation (#pcdata)>
<!ELEMENT practice (#pcdata)>
<!ELEMENT performance (#pcdata)>
```

FIGURE. 1

THE SYLLABUS DTD IN XML

Presentation, practice, and performance, defined in the last three lines of the above figure, represent items 4, 6, and 8 from Gagne's Events, and when applied will be composed of filenames and URLs representing the <u>content</u> of the tutorial. The files may be created by any application as long as they may be viewed from a browser. Hence, instructors are given the freedom to use software with which they feel most comfortable.

Given the DTD, the authoring software will utilize it to organize all the reference files and URLs the author wishes to use into one *Syllabus* file (an XML document) which contains the syntax information as well as the filenames (including full path information) and URLs.

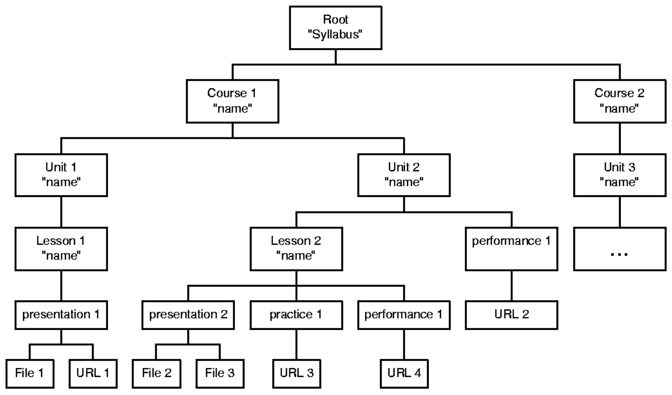

FIGURE. 2
THE SYLLABUS DTD AS A TREE

Figure 2 shows (in the form of a tree) an application of the DTD to a Syllabus for two Courses. Course 1 has two Units, Unit 2 has both a Lesson (#2) and a performance (#1). Note the recursive structure of the tree, and that leaves are the files / URLs containing the content of the tutorial.

PROGRAMMING LANGUAGE

While Sun Microsystems currently controls the overall direction of Java, the use of Java on the Internet is widespread and does not appear to be slowing. Java can lend itself to the idea of an open programming language due to its inherent object-oriented nature. Developers only need to develop application-programming interfaces (APIs), which can then be distributed to other people in the development community.

Since Java and XML are generally accepted in the Internet community and are extremely flexible to use, they are a good choice for our document definition and programming languages.

CROSS-PLATFORM

From its inception, the Internet has connected and resided on different types of hardware, from Macs to PCs to Cray computers. The need for a cross-platform solution can easily be seen as a requirement for distance learning and military personnel in the field, as well as for the variety of computers now in our public schools.

Once a cross-platform solution is created, it automatically has built-in flexibility since the software is able to run on any machine with a browser. This saves time and energy for the software's author since the need to write custom applications for different platforms is eliminated.

Java was created specifically to be cross-platform. This ability is accomplished with a Java Virtual Machine (JVM) on each platform on which we wish to run Java. Currently, JVMs are written in a native language for a specific platform and are then executed either as a stand-alone application or from within a web browser.

The user of a Java program only requires an installed JVM or a web browser (which already has JVM installed) to run a Java program. Java separates the idea of a stand-alone program, which is executed from a JVM, and that of a Java applet, which is executed through a web browser or other viewing program. From an end-user's perspective this separation does not matter since the code will be distributed in some form of packaging, rather than as raw code. Hence, use of Java satisfies (as much as is currently possible) the cross-platform issue.

SEPARATION OF CONTENT FROM DELIVERY

Once the content has been separated from the delivery mechanism, the creation of educational material can be done more cheaply and with less development time than if it were incorporated into the software itself. Application programmers can modify existing delivery mechanisms to provide increased functionality, while instructors can develop and share content material. Owing to the XML protocol, instructors can easily develop new tutorials that will work in a given application, or they can modify already existing tutorials.

XML is used to detail the structure and syntax rules for the creation of the content file. This content file is what allows the delivery software to correctly parse the output of the authoring software. The delivery mechanism must then be able to parse the XML by using an existing validating or non-validating XML parser and present the material in an appealing manner in either a Java application (stand-alone) or as an applet (for people with web browsers).

CREATION, EDITING, & DELIVERY MECHANISMS

Application users need the ability to compose and modify content, which is easiest for most users with a graphical user interface, or GUI. Our creation and editing tool is written as a Java application that provides the user a GUI through the existing Java Swing APIs (Figure 3). This Java application produces an XML file representing the composition of the content (i.e., the filenames and URLs).

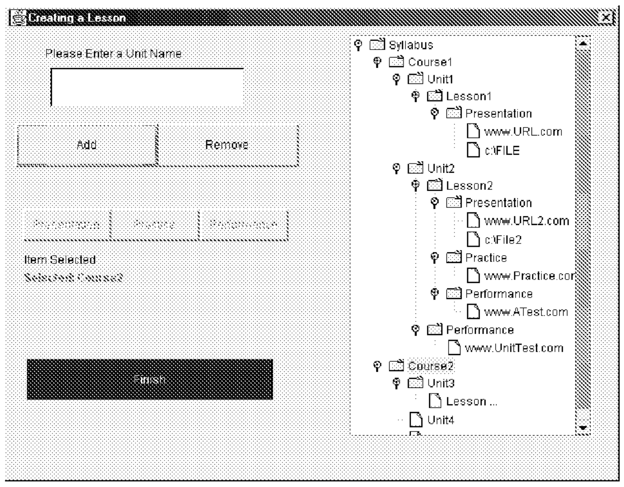

FIGURE. 3
CREATING AND EDITING A SYLLABUS.

The delivery mechanism is also written in Java, and can be utilized as an applet (using a browser either locally or over the Internet), or as an application (stand-alone software). An example is shown in Figure 4.

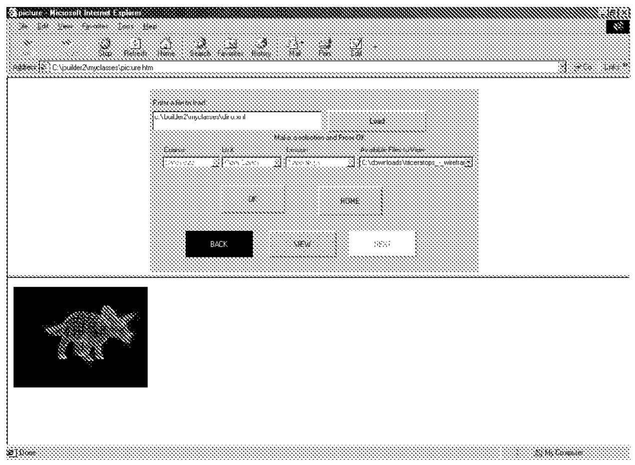

FIGURE. 4
VIEWING THE TUTORIAL

CONCLUSION

By successfully meeting each of the criteria – using an open standard, providing cross-platform compatibility, separating content from delivery, and providing an authoring and viewing environment – we have shown the feasibility of an open protocol for educational material presented either over the Internet, over a local network, or residing on a local machine. Further development in this area is needed to stem the rising tide of proprietary, expensive, and bloated educational software, and to fill the niche created by the legacy computers which are currently under-utilized. It is an idea whose time is at hand.

REFERENCES

[1] The White House, Office of the Vice President, *Information Technology for the Twenty-first Century: A Bold Investment in America's Future* [online]. www.pub.whitehouse.gov/uri-res/I2R?urn:pdi://oma.eop.gov.us/1999/1/25/12.text.1, 1998.

[2] Soloway, E., "No One is Making Money in Educational Software," *Communications of the ACM*, Vol. 41, No. 2, 1998, 11-15.

[3] Känel, j., Givler, J.S., Leiba, B., and Segmuller, W., "Internet Messaging Frameworks," *IBM Systems Journal*, Vol. 37, No. 1, 1998, 4-18.

[4] Edwards, D.M., and Hardman, L., "'Lost in Hyperspace': Cognitive Mapping and Navigation in a Hypertext Environment," *Hypertext: Theory into Practice*, 1989, 105-125.

[5] Gagne, R., Briggs, L., and Wager, W. *Principles of Instructional Design*. Holt, Rinehart, and Winston, 1988.

[6] York, B., *Course Content Authoring and Sequencing Using XML*, Masters Thesis, Computer Science Dept., Texas Tech University, 1999.